Successive Approximation ADC Datasheet SAADC V 1.10

Copyright © 2002-2014 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY7C603xx	0	1	1	237	4	1

See the application note “Analog - ADC Selection” AN2239 for other converters.

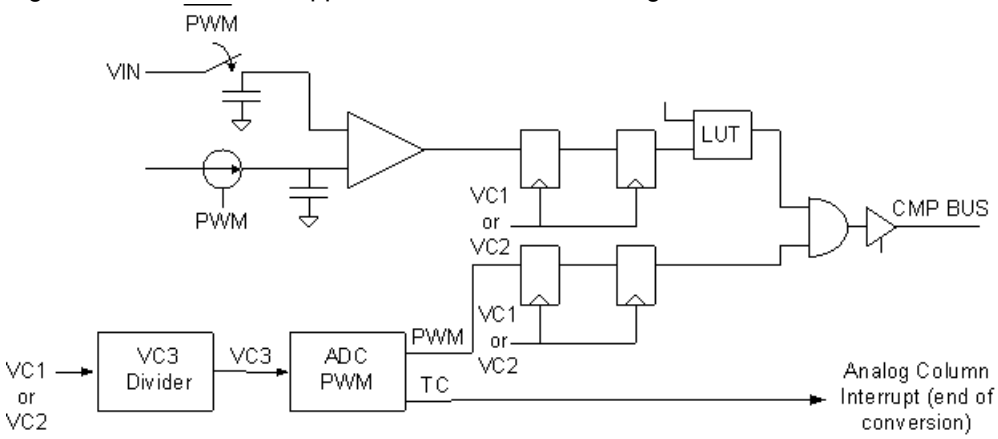
Features and Overview

- 8-bit resolution
- Sample rates up to 100 sps
- Input range depends on CPU Clock

The SAADC implements a simple ADC which does not require a Digital Block to operate.

Note The SAADC has 8 bits of resolution with 1% error if placed in analog column 0. When placed in analog column 1 the error is 4%.

Figure 1. Successive Approximation ADC Block Diagram



Functional Description

The SAADC User Module implements a successive approximation A/D converter that generates an 8-bit output. The user module gathers one sample through a provided API function call. The time it takes for a single sample to be collected is non-deterministic and depends a great deal on the input voltage being measured and the CPU speed.

The SAADC is constructed with the following PSoC resources:

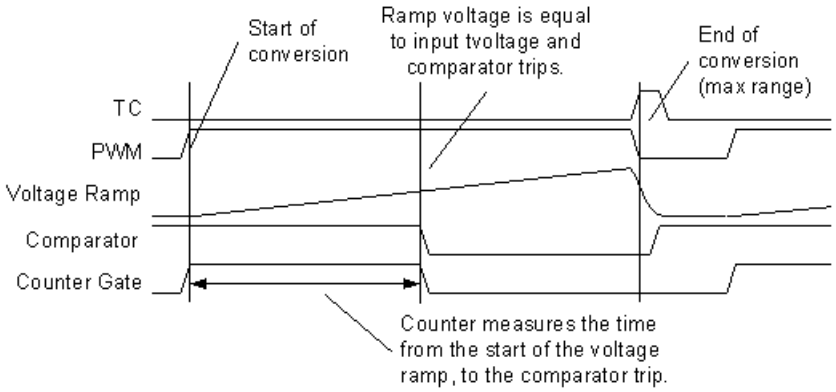
- An Analog CT Block, configured as a comparator
- An Analog SC Block, configured as a ramp generator
- A dedicated PWM used for ADC timing

- The VC3 clock source

As shown in the block diagram, the core of the conversion algorithm involves a current source, an integrating capacitor, a comparator and the VC3 clock source. The goal of the successive approximation algorithm is to find the value of the VC3 divider which causes the PWM high time to equal the time it takes for the comparator to trip. This is accomplished by varying the VC3 divider so that the PWM high time - seen in the SAADC Conversion Timing Diagram below matches the comparator time. The Timing Diagram below is repeated eight times with the PWM high time moving incrementally closer to matching the comparator output. With every iteration the current source is activated and a linear voltage ramp is generated on the capacitor. The capacitor voltage is an input to the analog comparator circuit, the other input of which is the analog input voltage being converted. At the end of the conversion output of the comparator is tested to determine whether the comparator was tripped, if it did the PWM high time is reduced incrementally, if it didn't the time it is increased.

The basic conversion waveform is show in the diagram below. The low time of the PWM is designed to allow the capacitor to discharge. The Terminal Count of the PWM is used as an interrupt to read the results of the conversion. Bit 7 of ADC_CR is set high if the conversion is greater than 255 counts. This PWM generator is clocked from VC3 only and has a limited selection of high and low times.

Figure 2. SAADC Conversion Timing Diagram



Even though there are two analog columns available, there is only one ADCPWM circuit available and it is shared by both columns which means that only one ADC can be placed in a project.

Calibration

The ADC must be calibrated before use. Each column has an ADC capacitor trim register for this purpose. This register controls the capacitor value that determines the slope of the ADC voltage ramp. The CAPVAL [7:0] bits of the ADC_TR register are used for calibration. Before the converter can be used, the capacitor array must be set to the correct value. The goal of this calibration process is to tune the ramp time (slope) such that a full-scale ADC input value results in a full scale ADC code. This is accomplished by matching the ramp time to the desired full-scale conversion period.

The user module API includes a routine called SAADC_bCal that executes a calibration algorithm that trims the ADC_TR register based on a reference voltage and the value that the user identifies as the expected full-scale result for that voltage. The SAADC_bCal routine accepts two arguments - the expected full-scale conversion result for the reference voltage and the source of the reference voltage.

The Equation 1 below shows how the expected conversion result can be calculated based on the reference voltage and the full-scale voltage.

Equation 1

$$ExpectedCode = \left(\frac{V_{CalRef}}{V_{FullScale}}\right)255$$

The internal Bandgap voltage (AGND) which provides a 1.3V signal is a convenient calibration reference. If Vdd is at 5V and the ADC is powered to FULL_RANGE then the expected conversion result for the 1.3V reference is 66 as shown in the calculation below.

Equation 2

$$ExpectedCode = \left(\frac{1.3V}{5V}\right)255 = 66.3$$

Alternatively an external voltage can be placed on any of the pins and then specified as the source for the calibration reference. The bCal routine will automatically switch its input to the selected reference source and then return to the original value upon exit.

The reference voltage has to be chose carefully since the range of the ADC is related to its CPU clock speed. If the CPU clock is less than 6MHz then the 1.3V bandgap can't be used as the reference since the range of the ADC will not be able to measure any voltage below 2.0V. The return value of the SAADC_bCal routine specifies the nearest possible value to which it was possible to calibrate the ADC relative to the expected conversion value. If there was a mismatch then the returned value can be used to perform offset error compensation if more accuracy is required.

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the following table, TA = 25°C, Vdd = 5.0V, Low Range, calibrated to VBG (1.3V), sample rates of 5.2 ksps, and a data clock of 2 MHz, unless otherwise noted.

Table 1. 5.0V SAADC DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	0.4 to Vdd-1V		
Input Capacitance[1]		--	pF	
Input Impedance	1/(C*Clk)	--	Ω	
Resolution	--	8	Bits	
Sample Rate	--	.5 to 8.8	ksps	
SNR		--	dB	
DC Accuracy				
INL		--	LSB	
DNL		--	LSB	

Parameter	Typical	Limit	Units	Conditions and Notes
Offset Error		--	mV	
Gain Error			mV	
Operating Current			uA	
Data Clock	--	0.24 to 2.4	MHz	Input to digital blocks and analog column clock

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the following table, TA = 25°C, Vdd = 3.3V, Low Range, calibrated to VBG (1.3V), sample rates of 5.2 ksps, and a data clock of 2 MHz, unless otherwise noted.

Table 2. 3.3V SAADC DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	0.4 to Vdd-1V		
Input Capacitance[1]		--	pF	
Input Impedance	1/(C*Clk)	--	Ω	
Resolution	--	8	Bits	
Sample Rate	--	.5 to 8.8	ksps	
SNR		--	dB	
DC Accuracy				
INL		--	LSB	
DNL		--	LSB	
Offset Error		--	mV	
Gain Error			mV	
Operating Current			uA	
Data Clock	--	0.24 to 2.4	MHz	Input to digital blocks and analog column clock

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the following table, TA = 25°C, Vdd = 2.7V, Low Range, calibrated to VBG(1.3V), sample rates of 5.2 ksps, and a data clock of 2 MHz, unless otherwise noted.

Table 3. 2.7 SAADC DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd-1V		
Input Capacitance[1]		--	pF	
Input Impedance	1/(C*Clk)	--	Ω	
Resolution	--	8	Bits	
Sample Rate	--	.5 to 8.8	ksps	
SNR		--	dB	
DC Accuracy				
INL		--	LSB	
DNL		--	LSB	
Offset Error		--	mV	
Gain Error			mV	
Operating Current			uA	
Data Clock	--	0.24 to 2.4	MHz	Input to digital blocks and analog column clock

Placement

The Analog blocks will be placed in the same column and will use the comparator bus. There are two columns available on the CY7C603xx family of devices. One and only one SAADC User Module can be used within a single project.

Parameters and Resources

Input

This parameter detmines the signal source for the A/D conversion. After the analog blocks are placed this parameter can be configured to one of the allowed values. A signal on a pin can be connected through the use of the Analog Column Input Mux or the Analog Mux Bus. The other options available are to connect to the neighboring column or to the Analog Reference VBG.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include” files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile” policy was selected for efficiency reasons. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

SAADC_Start

Description:

Performs all required initialization for this user module and turns on power for the ADC block.

C Prototype:

```
void  SAADC_Start(BYTE bRange)
```

Assembly:

```
mov   A,SAADC_FULLRANGE
call  ACD8_Start
```

Parameters:

bRange: Specifies the measurable input range for the ADC. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Description
SAADC_LOWRANGE	1	The input signal can be measured from Vss to Vdd-1V
SAADC_FULLRANGE	3	The input signal can be measured from Vss to Vdd

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SAADC_Stop

Description:

Turns off power to the ADC block.

C Prototype:

```
void  SAADC_Stop (void)
```

Assembly:

```
call  ACD8_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

SAADC_bCal

Description:

Calibrates the ADC by trimming the Capacitor in the SC block which in turn varies the slope of the reference ramp used in the conversion. The SAADC input must be set to a user_selected fixed reference voltage to which to calibrate. This function must be called after calling Start and before calling StartADC. Refer to the Functional Description for more information about calibration and about this function.

C Prototype:

```
BYTE   SAADC_bCal(BYTE bCalVal, BYTE bCalInput)
```

Assembly:

```
mov    A, [bCalVal]
mov    X, SAADC_CAL_P0_5
call   ADC_bCal
```

Parameters:

bCalVal: This number is value that is expected given the reference voltage to which the input is set. The SAADC_bCal routine will trim the capacitor in the SC block until the result is equal to or as close to as the bCalVal as possible.

bCalInput: This value specifies the source of the calibration input. This can be an internal reference such as AGND which provides a 1.3V signal or an external reference available on one of the pins. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Description
SAADC_CAL_VBG	The signal used for calibration will come from VBG the analog reference
SAADC_CAL_AMUXBUS	The signal used for calibration will come from a pin connected to AMUXBUS
SAADC_CAL_P0_0	The signal used for calibration will come from Pin 0.0 (Not available for Col 0)
SAADC_CAL_P0_1	The signal used for calibration will come from Pin 0.1

Symbolic Name	Description
SAADC_CAL_P0_2	The signal used for calibration will come from Pin 0.2 (Not available for Col 0)
SAADC_CAL_P0_3	The signal used for calibration will come from Pin 0.3
SAADC_CAL_P0_4	The signal used for calibration will come from Pin 0.4 (Not available for Col 0)
SAADC_CAL_P0_5	The signal used for calibration will come from Pin 0.5
SAADC_CAL_P0_6	The signal used for calibration will come from Pin 0.6 (Not available for Col 0)
SAADC_CAL_P0_7	The signal used for calibration will come from Pin 0.7

Return Value:

Returns one byte indicating the nearest result to the bCalVal that was possible.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

SAADC_StartADC

Description:

Turns on the ADC and runs it continuously. Global interrupts must be enabled for the ADC to function.

C Prototype:

```
void  SAADC_StartADC(void)
```

Assembly:

```
call  ACD8_StartADC
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

SAADC_StopADC

Description:

Immediately stops the ADC.

C Prototype:

```
void  SAADC_StopADC (void)
```

Assembly:

```
call  SAADC_StopADC
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

SAADC_fIsDataAvailable

Description:

Checks the status of the ADC.

C Prototype:

```
BYTE  SAADC_fIsDataAvailable(void)
```

Assembly:

```
call  SAADC_fIsDataAvailable
```

Parameters:

None

Return Value:

Returns a non-zero value if data has been converted and is ready to read.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

SAADC_ClearFlag

Description:

Resets the data-available flag.

C Prototype:

```
void  SAADC_ClearFlag(void)
```

Assembly:

```
call  SAADC_ClearFlag
```

Parameters:

None

Return Value:

None

Side Effect:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

SAADC_bGetData

Description:

Returns converted data. SAADC_fIsDataAvailable() should be called to verify that the data sample is ready.

C Prototype:

```
BYTE  SAADC_bGetData(void)
```

Assembly:

```
call  SAADC_bGetData
```

Parameters:

None

Return Value:

Returns the converted data sample.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

SAADC_bGetDataClearFlag

Description:

Returns converted data and resets the data available flag. SAADC_fIsDataAvailable() should be called to verify that the data sample is ready.

C Prototype:

```
BYTE  SAADC_bGetDataClearFlag(void)
```

Assembly:

```
call  SAADC_bGetDataClearFlag
```

Parameters:

None

Return Value:

Returns the converted data sample.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Sample Firmware Source Code

The sample code below polls the Flag register and sends the data to a routine that will shift the data out one of the I/O pins.

```
;;; Sample Code for the SAADC
;;; Continuously Sample input voltage
;;;
include "m8c.inc"               ; part specific constants and macros
include "PSoCAPI.inc"           ; PSoC API definitions for all User Modules

export _main

_main:
   M8C_EnableGInt               ; enable global interrupts
   mov  A,SAADC_FULLRANGE
   call SAADC_Start

   mov  A, 42h                  ; Set the calibration value to 42h
   mov  X, SAADC_CAL_VBG        ; Set the calibration source to AGND
   call SAADC_bCal              ; Calibrate the ADC to 1.3V = 42h

   call SAADC_StartADC
wait:
   call SAADC_fIsDataAvailable  ; poll flag
   jz wait

   call SAADC_bGetDataClearFlag  ; retrieve the data and reset flag
   ;; call shift_it_out          ; (user provided) send data to output pin
   jmp wait
```

The same project written in C is:

```
//-------------------------------------------------------------------
// Sample C Code for the SAADC
// Continuously Sample input voltage
//-------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

BYTE bData;
void main(void)
{
   M8C_EnableGInt;

   SAADC_Start(SAADC_FULLRANGE);     // Start the User Module

   SAADC_bCal(0x42, SAADC_CAL_VBG); // Calibrate the ADC so 1.3V = 0x42
```

```
   SAADC_StartADC();                // Start

// Begin sampling
while(1)    {
    while(0 == SAADC_fIsDataAvailable()) { }
    bData = SAADC_bGetDataClearFlag();
   // Add user code here to Display the result
}
}
```

Configuration Registers

The input multiplexer determines what signal is digitized.

Table 4. Block ADC: Register ACE_CR1

Bit	7	6	5	4	3	2	1	0
Value	0	1	1	0	1	Input		

The Input bits determine the signal source for the A/D conversion. This value is set by the Input parameter and can be changed in order to provide alternative reference voltage for calibration.

Table 5. Block ADC: Register ACE_CR2

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	1	Enable

The Enable bit is used to start the user module.

Table 6. Block RAMP: Register ASE_CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 7. Block RAMP Register ADC_CR

Bit	7	6	5	4	3	2	1	0
Value	CMPST	1	1	0	0	1	0	Enable

The Enable bit is used to start the user module. The CMPST bit is a read-only bit that indicates whether the comparator tripped during the conversion period or not. This is used to determine whether an over range condition has occurred or not.

Table 8. Block RAMP: Register ADC_TR

Bit	7	6	5	4	3	2	1	0
Value	Capacitor Trim Value							

The Capacitor Trim Value is set during the bCal function. This 8-bit value sets the value of the adjustable capacitor and thus the slope of the reference ramp used by the ADC.

Version History

Version	Originator	Description
1.0	DHA	Initial version.
1.10	MYKZ	Corrected method of clearing posted interrupts.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2002-2014 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.